PCIE8T Dinigroup Board Family Full Design User Interface Manual
2/19/2008
Rev 1.3
Written By: The Dini Group

This document corresponds to PCIE FPGA version 0x00010058

Table of Contents

1 Overview 4
1.1 Purpose 4
1.2 Board Architecture overview 4
1.3 PCI-E FPGA types 5
1.3.1 User Mode Details 5
1.3.2 PIPE Mode Details 5
1.3.3 TLP Mode Details 6
1.3.4 Full Mode Details 6
2 Full Mode PCI-E Side Details 6
2.1 An Overview of DMA 6
2.2 Specifics of Dinigroup's DMA Engines 6
2.3 DMA related Bar0 Registers 7
2.4 DMA Descriptor Format 11
3 Full Mode User FPGA Side Details 12
3.1 Pin Details 12
3.2 Backend Signals 13
4 Future Features yet to be Implemented or Tested 17
4.1 Target Features 17
4.2 DMA To-Host Features 17

Table of Tables

Table 1: Change Log 3
Table 2: Bar0 Address Map 7
Table 3: Descriptor Format 11
Table 4: Backend Clocks, Resets 13
Table 5: PCI-E Signals (to/from FPGA pins) 14
Table 6: User Signals 15

Table of Figures

Figure 1: PCI-E FPGA Major Bus Connections 5
Figure 2: Clocking, Reset/Lock signals 13

Table 1: Change Log

Revision	Name	Details
1.1	John Perry	Initial
1.2	John Perry	Added all of bar0 registers.
1.3	John Perry	Added Interrupt, RS232 control registers

1 Overview

1.1 Purpose

This document describes how to interface to the Dinigroup Boards that use FPGA transceivers for implementing PCI-E. We describe how to interface to the board from the host side (over the PCI-E connector using C code), and also how to interface to the PCI-E FPGA from the FPGA side ("backend", with the verilog the user puts into User FPGA A).

The interface that is described in this document is the "full" design, which includes Bar memory space and DMA engines. There exist other FPGA designs that The Dini Group provides for the PCI-E FPGA, that are not discussed in detail in this document.

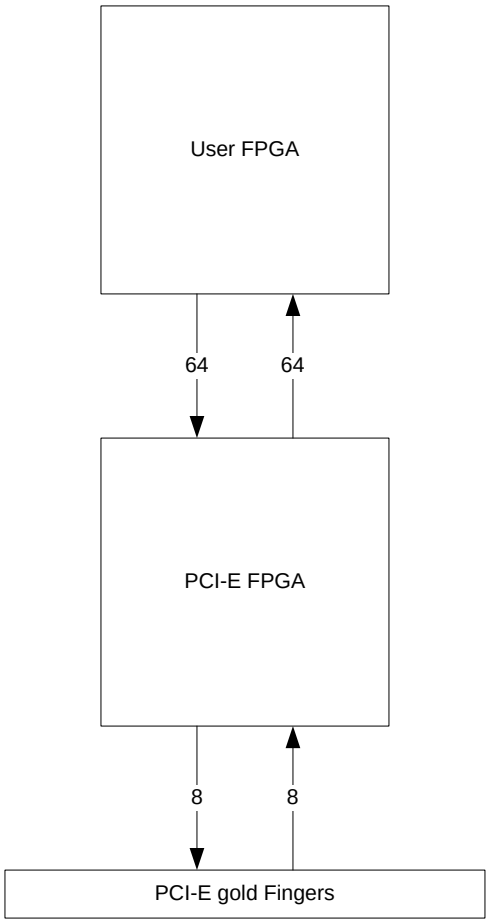
1.2 Board Architecture overview

There are 2 chips that are described in this document. The first is the PCI-E FPGA, which connects to the PCI Express gold fingers and the user FPGA. This FPGA acts as a PCI-E controller for data transfers to and from the board.

The second chip is the User FPGA, which is generally a very large FPGA that the user puts their logic into. This FPGA communicates with the PCI-E FPGA though a 64-bit unidirectional bus (64-bit bus in each direction). This User FPGA has a lot of other connections, which are beyond the scope of this document, and will vary from board to board.

Figure 1 shows the "large" bus connections between the PCI-E FPGA and the User FPGA.

Figure 1: PCI-E FPGA Major Bus Connections



Board Architecture 2, \\dncvs\FPGA\proprietary\pcie_v5t\Documents\Drawings\pcie_v5t_9kboards_architecture.vsd

1.3 PCI-E FPGA types

This section is a brief overview of the different PCI-E FPGA versions available on a DiniGroup Board.

The PCI-E FPGA has 4 different modes (and different bitfiles) that we must distinguish. The four modes are:

1) User Mode. The user can program whatever they want into the PCI-E FPGA.
2) PIPE Slowdown Mode. The User FPGA gets a PIPE interface at any clock frequency.
3) TLP Mode. The User FPGA gets TLP packets, at any clock frequency.
4) Full Mode. The PCI-E FPGA contains DMA engines and supports Bar access. The User FPGA uses a FIFO interface.

1.3.1 User Mode Details

The user can do whatever they want with the PCI-E FPGA. The Dini Group provides some example verilog to assist in some common tasks, such as programming the clock generator chip over I2C.

1.3.2 PIPE Mode Details

The PCI-E FPGA presents roughly the PIPE interface to the user FPGA. There are not enough pins to do a full 8-lane PIPE interface, so some pins need to be multiplexed. We provide a file that, when

included in the User FPGA, recreates the full PIPE interface, with some signals not updating immediately because of the pin multiplexing. The User FPGA supplies the clock for interface between the FPGAs, allowing emulation of a PIPE design at slower than 250 MHz.

1.3.3 TLP Mode Details

TLPs are sent between the two FPGAs at a user defined frequency. The PCI-E FPGA contains FIFOs to convert from the 250MHz PCI-E clock domain to the user clock domain. There are status signals for almost full and other FIFO information. Some pins are also used for PCI-E information such as credits available.

For TLP and Full modes, if the User FPGA is not ready (not configured, clocks not locked, etc.), the PCI-E portion of the design will not send transactions to the User FPGA.

1.3.4 Full Mode Details

The Full Mode contains 2 bidirectional DMA engines which can transfer data from-host, to-host, or both. Descriptors must be stored contiguously (for each engine) in host memory.

Both DMA engines support "Posted" DMA to-host and from-host transfers. That is, the DMA engine does not need a descriptor. Instead, the User FPGA simply provides a physical address followed by a bunch of data, and the DMA engine does the to-host transfer. Or, the user FPGA provides a physical address, a board address, and a length, and the DMA engine will fetch the data and return it to that board address.

Bar 0 registers allow the host to setup the DMA descriptor addresses and other DMA settings.

Other Bar 0 registers include setting the rocket IO synthesizer (Si5326), sending and receiving RS232 data, and reading the fan tachometer value.

Communication with the User FPGA is accomplished with an easy-to-use FIFO interface. A few verilog files, provided by The Dini Group, are required in the user FPGA.

2 Full Mode PCI-E Side Details

2.1 An Overview of DMA

DMA engines move data. DMA is a method of moving data without using a CPU to do the reads and writes. It also allows for bursting of data, which is usually not possible with CPU reads and writes.

A DMA engines uses descriptors to tell it what to do. A descriptor contains a length, source address, and destination address. It generally also contains flags/status to do different variants of data transfer, and to report back success or failure. The DMA engine uses this information to execute read or write requests, to move data from one place to another. The DMA engine also needs to know how to get descriptors and how to tell if they contain valid data or not.

2.2 Specifics of Dinigroup's DMA Engines

Our DMA engines use host memory to get descriptors. We use a valid bit in the descriptor to tell if the descriptor is valid or not. Software sets the valid bit to 1 when the 4 dword descriptor contains valid data. Hardware sets the valid bit to 0 when hardware is done moving the data. The descriptors in host

memory are stored in a contiguous circular queue. Software simply tells the dma engine the base address and size mask of the circular queue, and then hardware has enough information to fetch descriptors and process them. It is the responsibility of software to make sure that the tail of the queue is never overwritten.

Note that all addresses given to the DMA engine (both the descriptor queue base address and the pci-e addresses in the descriptors) must be physical addresses. The DMA engine has no means to translate logical addresses to physical addresses.

Queue terminology: Head is where the software writes the next valid descriptor. Tail is where hardware clears the descriptor when the transfer is complete. Top is the base address of the queue. Bottom is the last valid address in the queue, where software must "loop", so Bottom + 1 == Top.

2.3 DMA related Bar0 Registers

Here is the Address map:

Bar0: 0x0-0x1ff: PCI-E FPGA registers, rest is Configuration FPGA registers (8MB)

Bar1: 32-bit BAR, for User FPGA (8 MB)

Bar2-3: 64 bit BAR, for User FPGA (32MB)

Bar4-5: 64 bit BAR, for User FPGA (32MB)

These are the Bar0 Registers needed for DMA. R in first column means read only. W in first column means write only (reads back 0). R/W is read/writeable. These registers are accessible in the Full and TLP designs. The TLP design does not use the DMA registers, but for reusability of verilog they will still be accessible.

Table 2: Bar0 Address Map

Byte Addr	Name	Bit Definitions	Detailed Description
0x000 R	Version Number	[31:16] Major Rev. [15:0] Minor Rev.	Version Number of PCI-E FPGA design currently loaded. Current value is 0x0001_0043.
0x004 R	Date	YYYY_MMDD	Date that this design was created/built. Example value is 0x2007_0718
0x008 R	Design Type	ASCII of Bitstream Type.	Possible values are "Full" = 0x4675_6C6C or "TLP " = 0x544C_5020.
0x00C R/W	Clock Synth. Control	0: SCL 1: SDA OE 2: SDA RD 3: RSTN 4: enable control	I2C interface. Bits 0, 1, 3, 4 are read/write. Bit 2 is read only. SDA is open drain. It will float high. To drive low, set SDA_OE to 1. Set Bit 4 to 1 to control this bus (if 0, Si5326_controller in the verilog is controlling this

			bus).
0x010 R/W	Reset Control	0: Reset Config. 1: Reset User 2: Reset Config Value 3: Reset User Value 4: Reset_QN value	Bits 0 and 1 are read/write. Bits 2-4 are read only. Write a 1 to reset to drive reset to the user or the configuration FPGAs. Not self clearing (software must write a 0 sometime later). Bits 2 and 3 are used to tell if the reset is active or not (mostly for debugging). Bit 4 is used to tell if the soft_reset pushbutton is pressed (from Spartan FPGA).
0x014 R/W	Test Control	[0] RS232 Enable [1] LFSR Test Enable [2] LFSR Test Reset [31:3] LFSR Test Errors	Bits 2-0 are R/W. Bits 31-3 are R only. When high, enables RS232 loopback (sends RX back out on TX pin). When low, TX is tristated. BITS 2-0 SHOULD BE 0 EXCEPT WHEN TESTING PINS, BECAUSE FPGA A DRIVES RS232 ALSO, AND LFSR TEST BREAKS BAR0 TO CONFIG FPGA WHEN ENABLED.. The LFSR test tests the pins between the LXT and the Config FPGAs. LFSR Test Sequence: Write 0x6. Readback 0x6. Write 0x2. Readback (Expect:0x2). Wait a while (1-2 seconds). Read back, expect 0x2. If not 0x2, there are errors between the FPGAs. Write 0x0 to return to normal operation.
0x018 R/W	LED Control	[4:0] Software LED value [12:8] Software LED enable [20:16] LED value	Bits 4-0 and 12-8 are R/W. Bits 20-16 is read only. Bits 12:8 allow software to drive the LED when 1, with the value of 4:0. 20:16 is the actual value on the LED. Order of LEDs is (high->low bits): Yellow Activity, Red Loss of Signal, Green 8 link, 4 link, link.
0x01C R	Fan Tachometer	[15:0] Revolutions Per Second 16: Current Fan Tach Value [24:18] VRP pins [31:25] inverted VRN pins	We'll use a known clock to count revolutions, and store the value every second (24MHz or can we depend on 250MHz always being that speed?) Currently, this uses DMA clock, which isn't a known frequency, but triggers assuming a 125 MHz clock. This revolution counter is absurdly big (need 7 or 8 bits).

0x020 R/W	DMA0 Base Address0	[31:12] Base Address [11:0] 0x000	Lower 32 bit byte address of physical address where the DMA0 descriptor chain starts. This address must have the lower bytes cleared to match the DMA0 Address Mask register.
0x024 R/W	DMA0 Base Address1	[31:0] Base Address	Upper 32 bits of Base Address[63:0], to form a 64 bit address. Set to 0 if using 32 bit addressing.
0x028 R/W	DMA0 Address Mask	[19:12] Address Mask [11:0] 0xFFF	Address mask to indicate size of Descriptor list. Set some number of consecutive lower bits to 1. Set to the size of the list in bytes, minus 1. For example, for an 8KB descriptor list, set this to 8K-1=0x01FFF.
0x02C R/W	DMA0 Control	[9:0] re-read time in milliseconds 10: DMA enable 11: DMA clear 12: DMA Post Enable	Re-read time is the time between checks for a valid bit in the next descriptor in the list. This value can not be set to 0. If software tries to set this to zero, it will be set to one. Default value is 1. DMA enable enables the DMA engine. DMA clear clears the DMA engine, resetting the address counters to the base address and clearing any running transactions. Software must set this back to zero to bring the DMA engine out of "reset". DMA Post Enable means that the DMA engine will discard all to-host descriptors, and instead only respond to Posted transfers from the User FPGA or Configuration FPGA.
0x030 W	DMA0 Poll Immediate	0: Poll Immediate	Writing a 1 to this register causes the DMA engine to fetch the next descriptor from the host immediately (and not wait for the re-read timer to expire).
0x034 R	DMA0 Read Address	[31:0] Current Read Address	DEBUG INFO ONLY: The lower 32 bits of the current address being read, or the next address that will be read.
0x038 R	DMA0 Execute Address	[31:0] Current Execute Address	DEBUG INFO ONLY: The lower 32 bits of the address of the descriptor currently being worked on. Because of multiple reads pending, we could have subsequent descriptors also being worked on at the same time.
0x03C R	DMA0 Descriptor FIFO Info	[9:0] Descriptor FIFO empty count [16] Test Bit	DEBUG INFO ONLY: Number of elements (dwords or qwords?) in the descriptor FIFO. Note there could be more descriptors still pending in the system that have already been read out of this

		(DMA0 only)	FIFO. Test Bit should be 1 in normal operation.
0x040-0x05C	Repeat for DMA1		
0x060 R	DMA Clock Counter	[31:0] DMA Clock Counter	Counts every clock on the DMA clock domain (should be slightly slower than 250MHz, derived from clock from xilinx PCI-E core).
0x064 R	User Clock Counter	[31:0] User Clock Counter	Counts every clock on the User clock domain (should be whatever clock the user supplies (PCIE_PCLK_QP/N)).
0x068 R	Config Clock Counter	[31:0] Config Clock Counter	Counts every clock on the Config clock domain (should be 96 MHz, CLK_QL_Q through DCM).
0x06C R	48MHz Clock Counter	[31:0] 48 MHz Clock Counter	Counts every clock on the 48 MHz clock domain (should be 48 MHz (main bus clock), CLK_MB48_QP/N).
0x070 R	Ref Clock Counter	[31:0] Ref Clock Counter	Counts every clock on the Ref clock domain (should be 250MHz??, CLK_REF_QP/N).
0x074 R	GTP Clock Counter	[31:0] GTP Clock Counter	Counts every clock on the GTP clock domain (should be whatever U24 is set to drive out, CLK_GTP_QP/N).
0x078 R	EXT0 Clock Counter	[31:0] EXT0 Clock Counter	Counts every clock on the EXT0 clock domain (should be ?? MHz, CLK_EXT0_QP/N).
0x07C R	EXT1 Clock Counter	[31:0] EXT1 Clock Counter	Counts every clock on the EXT1 clock domain (should be ?? MHz, CLK_EXT1_QP/N).
0x080 R	G0 Clock Counter	[31:0] G0 Clock Counter	Counts every clock on the G0 clock domain (should be ?? MHz, CLK_G0_QP/N).
0x084 R	G1 Clock Counter	[31:0] G1 Clock Counter	Counts every clock on the G1 clock domain (should be ?? MHz, CLK_G1_QP/N).
0x088 R	G2 Clock Counter	[31:0] G2 Clock Counter	Counts every clock on the G2 clock domain (should be ?? MHz, CLK_G2_QP/N).
0x08C R	TP_Q Clock Counter	[31:0] TP_Q Clock Counter	Counts every clock on the TP_Q clock domain (should be ?? MHz, CLK_TP_QP).
0x090 R	CLKM Clock Counter	[31:0] CLKM Clock Counter	Counts every clock on the CLKM_Q clock domain (should be 24 MHz).
0x094 R	CFG Clock Counter	[31:0] CFG Clock Counter	Counts every clock on the CFG clock domain (should be ?? MHz (CFGCLK out of USR_ACCESS_VIRTEX5 primitive)).
0x098 R/W	Interrupt	[0] DMA0 To	In Legacy Interrupt mode, a value of 1 means that

		Host Interrupt [1] DMA0 From Host Interrupt [2] DMA1 To Host Interrupt [3] DMA1 From Host Interrupt [4] DMA0 Going Idle [5] DMA1 Going Idle	this interrupt event occurred. Software needs to write a 1 to the corresponding bit to clear the interrupt. In MSI interrupt mode, these bits are self-clearing once the MSI interrupt message is sent. The vector number send per interrupt depends on the number of vectors allocated to the card, and will be defined in a future version of the design.
0x9C R/W	Interrupt Mask	[5:0] Mask Control for Interrupt Bits	A value of 1 in the mask means that the corresponding interrupt source generates an interrupt over the PCI-E bus. A value of 0 means the corresponding interrupt bit doesn't create an interrupt event.
0xA0 R/W	RS232 Control	[0] RS232 Output Enable [1] RS232 Output Data [2] RS232 Input Data	Bit 0: R/W: Enables driving data on the RS232 TX line. 1 is "on", 0 is "off". Bit 1: R/W: Value driven on the RS232 TX line when bit 0 is 1. Bit 2: R only: Value of RS232 RX line.
0x0A4-0x1FC R/W	Block RAM	[31:0] Block RAM data	Read/Write "scratch pad" space.
0x200-0x7F_FFFC	Configuration FPGA Space	See Configuration FPGA doc.	See Configuration FPGA documentation.

2.4 DMA Descriptor Format

This is the DMA Descriptor Format.

Table 3: Descriptor Format

Dword Number	Bit Number	Name	Description
0	31	Valid	Bit indicating if the descriptor is valid. Software sets this to 1 when the descriptor is ready to be processed (Other dwords of the descriptor written, memory pointed to by PCI Address available), Hardware clears this to 0 when the descriptor has been processed completely. **Software may not modify ANYTHING in the descriptor once the valid bit has been set to 1.**

0	30	Direction	0 = From_Host (Move data from PCI Address to Board Address) 1 = To_Host (Move data from Board Address to PCI Address)
0	29	Generate Interrupt	If 1, hardware will interrupt software once this descriptor has been processed completely. Bit is ignored when Set Registers Type is 1.
0	28	Set Registers Type	Set to 1 to set High Bits of Board Address. Does not transfer any data. Used to access more than 16 GB of memory space on the user side. This sets distinct registers depending on the value of the Direction Bit (To_host and From_host can have different high bits of the board address).
0	27	Config FPGA Job	If 1, indicates to transfer data to/from Spartan FPGA (configuration FPGA), instead of User FPGA.
0	26	RSVD	Reserved/unused/undefined. Set to 0 for future backward compatibility.
0	25	Overflow	When set, User returned more data than expected.
0	24	Transfer Complete	Status bit (set by Hardware). To_Host direction only, indicates user FPGA indicated this is end of the data for this transfer.
0	23:0	Byte Length[25:2]	Valid values are 4 bytes (value of 1) to 64M-4 bytes. This length must indicate a contiguous block of memory in PCI address space and user address space.
1	31:0	Board Address[31:0]	Dword Address to read or write from on the board (user side address space).
2	31:0	PCI Address[31:2], 00	Physical address to read or write data in host memory. Note that the data must be address contiguous for this entire transfer.
3	31:0	PCI Address[63:32]	Upper 32 bits of PCI physical Address. If software is using 32 bit addressing, this must be set to 0.

3 Full Mode User FPGA Side Details

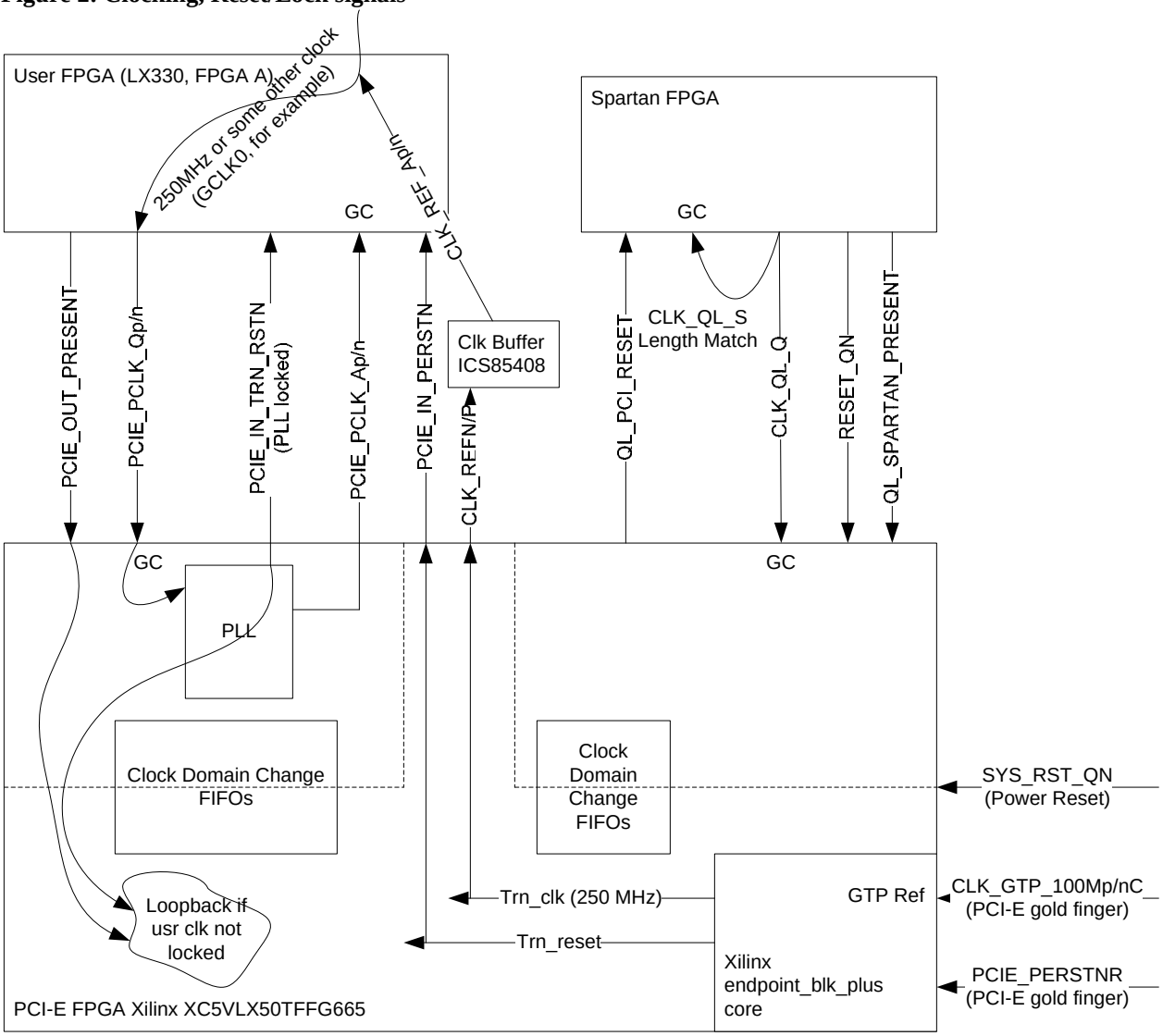
This section mainly describes the pcie_x8_user_interface.v module.

3.1 Pin Details

The user should use the reference design supplied by The Dini Group to handle pin assignments, clocks, resets, and other required functions. The user should never have to look at the FPGA to FPGA signals, because of the pcie_x8_user_interface module, but here are a couple details for background.

The user needs to supply a source-synchronous clock on PCIE_PCLK_Qp/n, as shown in the figure here.

Figure 2: Clocking, Reset/Lock signals



TLP/Full Mode Clock Diagram, \\dncvs\FPGA\proprietary\pcie_v5t\Documents\Drawings\pcie_v5t_9kboards_architecture.vsd

3.2 Backend Signals

Here we discuss the signals in and out of the pcie_x8_user_interface. The user must instantiate this module in order to interface with the PCI-E FPGA. Signals on both "sides" of this module are documented here, though the user is mainly concerned with the "User Signals". The "PCI-E Signals" are documented to give a sense of what information is on the bus between the two FPGAs. Direction is in relation to the pcie_x8_user_interface module.

The following signals are the resets and clocks.

Table 4: Backend Clocks, Resets

Signal Name	Direction	Description
Pcie_reset	Input	Main module reset. Held in reset when PCI-E FPGA clock is not locked,

		PCI-E PERSTN is asserted, trn_rstn is asserted, or resetn from the configuration FPGA is asserted (which includes soft reset push-button).
Pcie8t_qclk_out	Input	Main (user) clock
Pcie8t_qclk_in	Input	Clock for pin inputs (same frequency as pcie8t_qclk_out, but source synchronous with input pins from PCI-E FPGA). Connect to PCIE_PCLK_QP/N though IBUFGDS and BUFG.

The following signals are the "PCI-E Signals" on the "pin" side of the module. The inputs (_reg) need to be connected to the corresponding FPGA pin, through a flip flop using pcie8t_qclk_in. The outputs need to go through an output flip flop clocked with pcie8t_qclk_out, and then to the correspondingly named output pins.

Since these signals are not used by the user, extensive descriptions are not provided. "In" and "Out" in the signal names specify direction is relation to the User FPGA.

Table 5: PCI-E Signals (to/from FPGA pins)

Signal Name	Direction	Description
From Host Direction		
pcie_in_chan_reg[2:0]	Input	
pcie_in_d_reg[63:0]	Input	
pcie_in_sof_reg	Input	
pcie_in_eof_reg	Input	
pcie_in_info_reg[1:0]	Input	
pcie_in_tc_reg[1:0]	Input	
pcie_in_extra_reg[8:0]	Input	
pcie_un_extra_reg	Input	
pcie_in_valid_reg	Input	
pcie_in_all_valid_reg	Input	
pcie_out_almost_full_preff	Output	PCI-E FPGA stops sending data to User FPGA when this is high. Stops within 16 clocks of this going high.
To Host Direction		
pcie_out_chan_preff[2:0]	Output	
pcie_out_d_preff[63:0]	Output	
pcie_out_sof_preff	Output	

pcie_out_eof_preff	Output	
pcie_out_info_preff[1:0]	Output	
pcie_out_tc_preff[1:0]	Output	
pcie_out_extra_preff[6:0]	Output	
pcie_out_valid_preff	Output	
pcie_out_all_valid_preff	Output	
pcie_out_present_preff	Output	
pcie_in_almost_full_reg	Input	User FPGA stops sending data to PCI-E FPGA when this is high.

The following signals are the signals that the user should hook up to.

This following table describes the data on the User FPGA side, that the user is expected to use and respond to. Both directions for DMA are a FIFO interface, with a few bits of tag to indicate what type of data is in the going in or out of the FIFO. Thus, in the following table we mostly just document the tag values and the data that goes along with each type of tag.

Table 6: User Signals

Signal Name	Direction	Description
Target Interface		
target_address[31:0]	Output	Byte address (though bits 3 to 0 are always 0)
target_address_valid	Output	One clock cycle strobe that indicates the target_address is valid
target_write_data[63:0]	Output	Data.
target_write_be[7:0]	Output	Byte enables, 0 means data[7:0] is valid, etc.
target_bar[2:0]	Output	Valid for reads and writes. Indicates which bar is being accessed (bar number 1, 2, or 4). Onehot, so bit 0=1 means bar1, bit 1=1 means bar2-3, and bit 2=1 means bar4-5.
target_write_enable	Output	Indicates valid write data on this interface.
target_write_accept	Input	Accepts the valid write data. Data is “transferred” when target_write_accept and target_write_enable are active. If connecting to blockram, this signal can be tied high.
target_read_enable	Output	High on a target read request.
target_request_tag[3:0]	Output	Tag associated with this read request.
target_read_accept	Input	Acceptance of read request. Allows interface to “move on”

		before the read data is returned by the user. Target_read_enable and target_read_accept being high signals “transfer” of request. If connecting to blockram, this signal can be tied high.
target_read_data[63:0]	Input	Read Data to return to PCI-E.
target_read_data_tag[3:0]	Input	Tag that accompanies this data. This value must match the value provided on target_request_tag when target_read_enable was high.
target_read_data_valid	Input	Clock cycle pulse indicating read data is valid.
DMA0 Interface		
dma0_from_host_data[63:0]	Output	Address, length, or data, depending on the state of dma0_from_host_ctrl[5,0].
dma0_from_host_ctrl[7:0]	Output	Bit meanings: 5, 0 indicates type of data. 3:2 are dword enables (when bit 3 is 1, 63:32 is valid, when bit 2 is 1, 31:0 is valid) 4 indicates read request when 1, write “packet” when 0. Bits 1, 6, 7 reserved. Decode of 5,0: 01: 64 bit dword board address. (Add two zeros to the bottom of this data if you want to think of it as a byte address). 10: 24 bits of dword length [23:0]. Upper 40 bits are reserved. Also valid for write packets, even though the user doesn’t really need it (for reads, user must return this many dwords of data, properly aligned based on bit 0 of the dword board address). 00: Data! 11: Reserved/Undefined/Never Occurs.
dma0_from_host_valid	Output	Indicates valid data on data and ctrl signals.
dma0_from_host_advance	Input	This signal high and dma0_from_host_valid high indicates data “transfer”.
dma0_to_host_data[63:0]	Input	Data to return to PCI-E FPGA.
dma0_to_host_ctrl[7:0]	Input	Bit meanings:

		1:0 are dword enables (when bit 1 is 1, 63:32 is valid data, when bit 0 is 1, 31:0 is valid data). 3: Indicates last data for this read request. Bits 2, 4-7 are reserved.
dma0_to_host_valid	Input	When 1, data/ctrl is transferred into the module.
dma0_to_host_almost_full	Output	When high, user should stop writing data to the module soon.
DMA1 Interface		
Same as DMA0	Same	Same as DMA0, but '1' in the name instead of '0'. Separate interfaces for each DMA engine.

4 Future Features yet to be Implemented or Tested

4.1 Target Features

Byte enables haven't been tested yet, we have no reason to believe this won't work.

4.2 DMA To-Host Features

Flush feature: The user can indicate that it is done sending data back for the current read request, even though the user has not sent back the requested amount of data. The user can indicate this flush on a clock cycle that they are not sending back any other data.

Status: This might work (using last), but is untested. It is more likely to work to indicate last with the last data, and it is ok if this isn't the full request size.

Posted Mode: The user FPGA can initiate to-host and from-host DMA transactions. For to_host transactions, the user supplies pci-e address, length, and then length dwords of data. For from_host, the user supplies pci-e address, board address, and length. In order for this mode to work, we need to define more bits in dma0_to_host_ctrl[7:0] to allow this functionality (pci-e address phase, board address phase, length (plus control (wr/rd for example) phase).

Status: Not implemented yet.